

Efficiently Computing Piecewise Flat Embeddings for Data Clustering and Image Segmentation

Renee T. Meinhold, Tyler L. Hayes, and Nathan D. Cahill
Image Computing & Analysis Laboratory, School of Mathematical Sciences
Rochester Institute of Technology, Rochester, NY, USA
{rtm9271, tlh6792, ndcsma}@rit.edu

Abstract—Image segmentation is a popular area of research in computer vision that has many applications in automated image processing. A recent technique called *piecewise flat embeddings* (PFE) has been proposed for use in image segmentation; PFE transforms image pixel data into a lower dimensional representation where similar pixels are pulled close together and dissimilar pixels are pushed apart. This technique has shown promising results, but its original formulation is not computationally feasible for large images. We propose two improvements to the algorithm for computing PFE: first, we reformulate portions of the algorithm to enable various linear algebra operations to be performed in parallel; second, we propose utilizing an iterative linear solver (preconditioned conjugate gradient) to quickly solve a linear least-squares problem that occurs in the inner loop of a nested iteration. With these two computational improvements, we show on a publicly available image database that PFE can be sped up by an order of magnitude without sacrificing segmentation performance. Our results make this technique more practical for use on large data sets, not only for image segmentation, but for general data clustering problems.

Keywords—image segmentation; clustering; embedding.

I. INTRODUCTION

Image segmentation is a popular area of research in computer vision and machine learning, as it is necessary for many applications such as automated visual recognition of objects in videos and photos and semantic image understanding. A recent method called *piecewise flat embeddings* (PFE) [1] performs image segmentation by creating a new pixel data representation (the *embedding*) in which pixels that contain similar characteristics are close together, while dissimilar pixels are far apart. The main attribute that sets PFE apart from other embedding methods is the piecewise constant or “flat” nature of the embeddings that makes clustering the pixels much easier than with other embeddings.

The PFE method relies on representing the image as a graph, with each pixel representing a vertex, and with similarities between pixels modeled by weighted edges between the vertices. As opposed to the well-known Laplacian Eigenmaps (LE) algorithm [2] that computes embeddings by minimizing a weighted ℓ_2 -norm of the differences between points in the new embedding, PFE minimizes a weighted ℓ_1 -norm subject to orthogonality constraints. This makes the embedding more difficult to compute than in LE, which reduces to a simple generalized eigenvalue problem. Yu et al. [1] propose a numerical procedure to approximate the solution to PFE, involving a nested looping procedure that requires Bregman

iterations [3]. However, as described in [1], there are many computational limitations when solving the problem in this manner.

In this paper, we propose two improvements to the numerical procedure for approximating the PFE: reformulating various linear algebra operations defined in [1] to use more compact matrices that enable much lower memory requirements, and utilizing an iterative linear solver (preconditioned conjugate gradient) as opposed to a direct solver to enable PFE to be computed on much larger graphs.

II. PIECEWISE FLAT EMBEDDINGS - BACKGROUND

Suppose $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ is a set of data points in \mathbb{R}^m . Dimensionality reduction algorithms like PFE or LE aim to generate a new set of corresponding points $\mathcal{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_n\}$ in \mathbb{R}^d , where $d \ll m$, so that inter-point distance relationships are preserved. To describe the relationships between the points in \mathcal{X} , we define a graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ with vertex set $\mathcal{V} = \{v_1, \dots, v_n\}$ and edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, where each vertex v_i corresponds to the point \mathbf{x}_i , and where the edge between vertices v_i and v_j is assigned a weight $w_{i,j}$. One common way to assign weights is according to the *heat kernel*; i.e., $w_{i,j} = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma^2)$, where σ is a parameter that can be selected by the user. Points in \mathcal{X} that are nearby will correspond to edges in the graph with weights that approach one, whereas points in \mathcal{X} that are far apart will correspond to edges with weights that approach zero.

Computing the PFE can be done by solving the constrained minimization problem:

$$\min_{\mathbf{Y}} \sum_{i,j} w_{i,j} \|\mathbf{y}_i - \mathbf{y}_j\|_1 \quad \text{s.t. } \mathbf{Y}^T \mathbf{D} \mathbf{Y} = \mathbf{I} \quad , \quad (1)$$

where $\mathbf{Y} = [\mathbf{y}_1, \dots, \mathbf{y}_n]^T$ is the $n \times d$ matrix containing the new set of points in \mathbb{R}^d , \mathbf{W} is the $n \times n$ *weighted adjacency matrix* that contains the weights $w_{i,j}$ for each edge in the graph, and \mathbf{D} is the *degree matrix* of the graph (i.e., the diagonal matrix $\mathbf{D} = \text{diag}(\mathbf{d})$, with $d_i = \sum_j w_{i,j}$).

The orthogonality constraint $\mathbf{Y}^T \mathbf{D} \mathbf{Y} = \mathbf{I}$ is required so that we can avoid the trivial solution $\mathbf{Y} = \mathbf{0}$; however, it makes (1) impossible to solve analytically. A numerical procedure for approximating its solution is presented in [1]. This procedure relies on the Splitting Orthogonality Constraint

Algorithm 1 SOC Algorithm for Approximating (2)

```

1: procedure SOC( $\mathbf{W}, \mathbf{Y}^{(0)}$ )
2:    $\mathbf{D} \leftarrow \text{diag}(\mathbf{W}\mathbf{1})$ 
3:    $k = 0, \mathbf{P}^{(k)} \leftarrow \mathbf{D}^{1/2}\mathbf{Y}^{(k)}, \mathbf{B}^{(k)} \leftarrow \mathbf{0}_{n \times d}$ 
4:   repeat
5:      $\mathbf{Y}^{(k+1)} = \arg \min_{\mathbf{Y}} \left( \sum_{i,j} w_{i,j} \|\mathbf{Y}_i - \mathbf{Y}_j\|_1 + \right.$ 
6:        $\left. \frac{r}{2} \left\| \mathbf{D}^{1/2}\mathbf{Y} - \mathbf{P}^{(k)} + \mathbf{P}^{(k)} \right\|_2^2 \right)$ 
7:      $\mathbf{P}^{(k+1)} = \arg \min_{\mathbf{P}} \left\| \mathbf{P} - \left( \mathbf{D}^{1/2}\mathbf{Y}^{(k+1)} + \mathbf{B}^{(k)} \right) \right\|_2^2$ 
8:       s.t.  $\mathbf{P}^T \mathbf{P} = \mathbf{I}$ 
9:      $\mathbf{B}^{(k+1)} = \mathbf{B}^{(k)} + \mathbf{D}^{1/2}\mathbf{Y}^{(k+1)} - \mathbf{P}^{(k+1)}$ 
10:     $k \leftarrow k + 1$ 
11:  until convergence
12:  return  $\mathbf{Y}^{(k)}$ 
13: end procedure

```

(SOC) algorithm [4]. To carry out this procedure, Yu et al. [1] define $\mathbf{P} = \mathbf{D}^{1/2}\mathbf{Y}$ and restate (1) as:

$$\min_{\mathbf{Y}} \sum_{i,j} w_{i,j} \|\mathbf{y}_i - \mathbf{y}_j\|_1 \quad \text{s.t.} \quad \mathbf{D}^{1/2}\mathbf{Y} = \mathbf{P}, \quad \mathbf{P}^T \mathbf{P} = \mathbf{I}. \quad (2)$$

The SOC algorithm (Algorithm 1) approximates the solution to (2) by performing a Bregman iteration, as described in [4].

The update $\mathbf{P}^{(k+1)}$ (line 6 of Algorithm 1) has a closed-form solution described in [1] that relies on matrices computed in the previous step. That previous step in line 5 that updates $\mathbf{Y}^{(k+1)}$ is an ℓ_1 -norm minimization problem that can be solved by the Split Bregman algorithm [3], which transforms ℓ_1 -norm minimization problems into series of differentiable convex optimization problems.

To write the update $\mathbf{Y}^{(k+1)}$ in a manner that can be solved via Split Bregman, Yu et al. [1] concatenate the columns of the matrices $\mathbf{Y}^{(k)}$, $\mathbf{P}^{(k)}$, and $\mathbf{B}^{(k)}$ into the vectors $\mathbf{Y}_v^{(k)}$, $\mathbf{P}_v^{(k)}$, and $\mathbf{B}_v^{(k)}$ respectively. They then define an $(n(n-1)/2) \times n$ sparse matrix \mathbf{M} that contains only two nonzero entries per row. If the graph edges are ordered according to $\mathcal{E} = \{(v_{i_k}, v_{j_k}), k = 1, \dots, n(n-1)/2\}$, then the only nonzero entries in the k^{th} row of \mathbf{M} are defined to be $M_{ki} = w_{i_k, j_k}$ and $M_{kj} = -w_{i_k, j_k}$. Next, a $(dn(n-1)/2) \times (dn)$ matrix \mathbf{L} and a $(dn) \times (dn)$ matrix $\tilde{\mathbf{D}}$ are defined as follows:

$$\mathbf{L} = \mathbf{I}_{d \times d} \otimes \mathbf{M}, \quad (3)$$

$$\tilde{\mathbf{D}} = \mathbf{I}_{d \times d} \otimes \mathbf{D}, \quad (4)$$

where \otimes indicates Kronecker product. Using all of this notation allows step 5 in the SOC algorithm to be rewritten as:

$$\mathbf{Y}_v^{(k+1)} = \arg \min_{\mathbf{Y}_v} \|\mathbf{L}\mathbf{Y}_v\|_1 + \frac{r}{2} \left\| \tilde{\mathbf{D}}^{1/2}\mathbf{Y}_v - \mathbf{P}_v^{(k)} + \mathbf{B}_v^{(k)} \right\|_2^2, \quad (5)$$

which can then be solved via Split-Bregman [3], as described in Algorithm 2. The update $\mathbf{Y}_v^{(k, \ell+1)}$ in step 5 requires solving a linear least-squares problem.

Algorithm 2 Split Bregman Algorithm for Approximating (5)

```

1: procedure SPLITBREGMAN( $\mathbf{M}, \mathbf{D}, \mathbf{P}^{(k)}, \mathbf{B}^{(k)}$ )
2:   Construct  $\mathbf{L}$  and  $\tilde{\mathbf{D}}$  from (3)–(4).
3:    $\ell = 0, \mathbf{b}^\ell \leftarrow \mathbf{0}_{(dn(n-1)/2) \times 1}, \mathbf{d}^\ell \leftarrow \mathbf{0}_{(dn(n-1)/2) \times 1}$ 
4:   repeat
5:      $\mathbf{Y}_v^{(k, \ell+1)} = \arg \min_{\mathbf{Y}_v} \left( \frac{\lambda}{2} \|\mathbf{L}\mathbf{Y}_v + \mathbf{b}^\ell - \mathbf{d}^\ell\|_2^2 + \right.$ 
6:        $\left. + \frac{r}{2} \left\| \tilde{\mathbf{D}}^{1/2}\mathbf{Y}_v - \mathbf{P}_v^{(k)} + \mathbf{B}_v^{(k)} \right\|_2^2 \right)$ 
7:      $\mathbf{d}^{\ell+1} = \text{Shrink}(\mathbf{L}\mathbf{Y}_v^{(k, \ell+1)} + \mathbf{b}^\ell, 1/\lambda)$ 
8:      $\mathbf{b}^{\ell+1} = \mathbf{b}^\ell + \mathbf{L}\mathbf{Y}_v^{(k, \ell+1)} - \mathbf{d}^{\ell+1}$ 
9:      $\ell \leftarrow \ell + 1$ 
10:  until convergence
11:  return  $\mathbf{Y}^{(k, \ell)}$ 
12: end procedure
13: procedure SHRINK( $\mathbf{y}, \gamma$ )
14:    $z_i = \text{sign}(y_i) \cdot \max(|y_i| - \gamma, 0), i = 1, \dots, \text{numel}(\mathbf{z})$ 
15:  return  $\mathbf{z}$ 
16: end procedure

```

III. EFFICIENTLY COMPUTING PFE

Suppose we would like to apply the PFE method to segment a 128×128 pixel image into 64 segments, and we create a graph where each vertex corresponds to a single pixel, and the only edges having nonzero weights connect vertices representing pixels that are within each others' 4-nearest neighbors. The sparse weighted adjacency matrix \mathbf{W} would require approximately 0.5 MB to store using double precision floating point values, and the matrix \mathbf{Y} would require 8 MB. For the Split-Bregman algorithm to be applied, the sparse matrices \mathbf{M} , \mathbf{L} , and $\tilde{\mathbf{D}}$ need to be computed; \mathbf{M} will be $134,209,536 \times 16,384$ with 131,072 non-zero entries (1 MB), \mathbf{L} will be $8,589,410,304 \times 1,048,576$ with 8,388,608 non-zero entries (64 MB), and \mathbf{D} will be $1,048,576 \times 1,048,576$ with non-zero entries on the main diagonal (8 MB).

Even for this small image with modest neighborhood structure, the amount of memory needed to store these matrices is large. The simple step of multiplying \mathbf{L} by the vector \mathbf{Y}_v in every Split-Bregman iteration will require significant computational effort. What becomes computationally prohibitive is step 5 of Algorithm 2. The normal equations for this linear least squares problem are:

$$\left[\frac{\lambda}{2} \mathbf{L}^T \mathbf{L} + \frac{r}{2} \tilde{\mathbf{D}} \right] \mathbf{Y}_v^{(k, \ell+1)} = \frac{\lambda}{2} \mathbf{L}^T \mathbf{q}_1 + \frac{r}{2} \tilde{\mathbf{D}}^{1/2} \mathbf{q}_2, \quad (6)$$

where $\mathbf{q}_1 = \mathbf{d}^\ell - \mathbf{b}^\ell$ and $\mathbf{q}_2 = \mathbf{B}_v^{(k)} - \mathbf{P}_v^{(k)}$. It would be unwise to invert $\frac{\lambda}{2} \mathbf{L}^T \mathbf{L} + \frac{r}{2} \tilde{\mathbf{D}}$ to solve (6); for our example, the inverse would be a dense $1,048,576 \times 1,048,576$ matrix, requiring 8 TB of storage. Even attempting to use a direct solver for (6) would require too much memory to be feasible, because although $\frac{\lambda}{2} \mathbf{L}^T \mathbf{L} + \frac{r}{2} \tilde{\mathbf{D}}$ is sparse and banded, its bands are far away from the main diagonal.

In order to combat these problems in computing PFE, we first note that if we define the function $\text{vec} : \mathbb{R}^{s \times t} \rightarrow \mathbb{R}^{st}$ that "unwraps" a matrix $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_t]$ into the vector $\text{vec}(\mathbf{Z}) = [\mathbf{z}_1^T, \dots, \mathbf{z}_t^T]^T$, then we can write $\mathbf{Y}_v^{(k)} = \text{vec}(\mathbf{Y}^{(k)})$, $\mathbf{P}_v^{(k)} =$

$\text{vec}(\mathbf{P}^{(k)})$, and $\mathbf{B}_v^{(k)} = \text{vec}(\mathbf{B}^{(k)})$. Using this notation, we can write:

$$\mathbf{L}\mathbf{Y}_v = \text{vec}(\mathbf{M}\mathbf{Y}) \quad , \quad (7)$$

$$\tilde{\mathbf{D}}^{1/2}\mathbf{Y}_v = \text{vec}(\mathbf{D}^{1/2}\mathbf{Y}) \quad . \quad (8)$$

Eq. (7) allows us to rewrite steps 6–7 of the Split-Bregman algorithm as:

$$\mathbf{d}^{\ell+1} = \text{Shrink}\left(\text{vec}\left(\mathbf{M}\mathbf{Y}^{(k,\ell+1)}\right) + \mathbf{b}^\ell, 1/\lambda\right) \quad , \quad (9)$$

$$\mathbf{b}^{\ell+1} = \mathbf{b}^\ell + \text{vec}\left(\mathbf{M}\mathbf{Y}^{(k,\ell+1)}\right) - \mathbf{d}^{\ell+1} \quad , \quad (10)$$

which reduces the computation required for these steps by a factor of d . Furthermore, we can write:

$$\mathbf{L}^T\mathbf{L}\mathbf{Y}_v = \text{vec}(\mathbf{M}^T\mathbf{M}\mathbf{Y}) \quad , \quad (11)$$

$$\tilde{\mathbf{D}}\mathbf{Y}_v = \text{vec}(\mathbf{D}\mathbf{Y}) \quad , \quad (12)$$

and therefore, (6) can be expressed alternatively as:

$$\left[\frac{\lambda}{2}\mathbf{M}^T\mathbf{M} + \frac{r}{2}\mathbf{D}\right]\mathbf{Y}^{(k,\ell+1)} = \frac{\lambda}{2}\mathbf{M}^T\mathbf{Q}_1 + \frac{r}{2}\mathbf{D}^{1/2}\mathbf{Q}_2 \quad , \quad (13)$$

where $\mathbf{d}^\ell - \mathbf{b}^\ell = \text{vec}(\mathbf{Q}_1)$ and $\mathbf{Q}_2 = \mathbf{B}^{(k)} - \mathbf{P}^{(k)}$. This provides our first improvement in efficiency in computing PFE: instead of solving the single $(dn) \times (dn)$ system of equations (6), we *simultaneously* solve the d different $n \times n$ systems of equations in (13). In fact, if we replace steps 5, 6, and 7 in the Split-Bregman algorithm with (13), (9), and (10) respectively, we see that the large matrices \mathbf{L} and $\tilde{\mathbf{D}}$ never have to be explicitly formed.

Secondly, we note that even though we have reduced the memory requirements of the system matrix in (13) by a factor of d over (6), we found by experimentation that it is still infeasible to solve (13) by explicitly inverting the system matrix or by invoking a direct solver. For this reason, we approximate the solution to (13) via the preconditioned conjugate gradient (PCG) method [5] with an incomplete Cholesky preconditioner.

IV. EXPERIMENTS

In order to investigate how well our efficient PFE method performs on an image segmentation task, we replicate one of the experiments in [1] that uses the 200 training images from the BSDS500 dataset [6], computes PFE, and performs segmentation by k -means clustering on the embeddings. We follow the two-stage approach of [1] that first carries out the nested Bregman iteration (Stage I) and then relaxes the orthogonality constraint and only carries out the Split-Bregman algorithm (Stage II). In Stage I, a maximum of 10 SOC iterations and 5 Split-Bregman iterations are performed, and in Stage II, a maximum of 100 Split-Bregman iterations are performed. The parameters λ and r are selected as in [1].

To construct a weighted graph for each image, we assign a graph vertex to each pixel (after downsampling images by a factor of 4 in each dimension), and we only assign nonzero weights (according to the heat kernel applied to the RGB image data) to pixels that are in each other's 4-neighborhood. To initialize \mathbf{Y} , we use the Gaussian Mixture Model (GMM) approach outlined in [1]. We then follow Yu et al.'s *dynamic*

scheme that chooses the embedding dimension d that produces the best performance out of dimensions in the range of 5 to 25. Figure 1 shows some of the BSDS500 training images along with examples of segmentation results via our implementation of PFE for various choices of embedding dimension.

We use the three criteria outlined in the BSDS segmentation benchmark [6] to assess how similar each segmented image is to the ground truth segmentation that was manually created and is provided with the BSDS500 dataset. The criteria are: covering, which quantifies overlap between segmentations; probabilistic rand index (PRI), which quantifies the "compatibility" of segmentations; and variation of information (VI), which describes the average conditional entropy of two segmentations. Larger covering and PRI values indicate better performance, whereas smaller VI values indicate better performance. The results are shown in Table I, along with the results of Yu et al.'s PFE implementation and three other methods tested in [1]: normalized cuts (NCut), spectral clustering (SC), and weighted spectral clustering (WSC). As we can see, our efficient implementation of PFE yields similar covering and PRI values to the results reported in [1], but slightly worse VI values. However, the standard deviation of our VI performance measure was computed to be 0.67. If Yu et al. had a similar standard deviation in VI values (which was not reported in [1]), it is likely that the difference in VI values between our implementation and Yu's is not statistically significant.

Finally, in Figure 2, we show box-and-whisker plots of the time (in seconds) required to compute PFE for the 200 BSDS500 training images, for a range of embedding dimensions. All computations are done in MATLAB. As can be seen in Figure 2, our implementation typically requires anywhere from 0.5–2 minutes, with more time required for higher embedding dimension. Yu et al. [1] report an average computing time of 15 minutes per image; however, as of the time of this article, they have not released any code, making it impossible for us to do a direct comparison.

Method	Covering	PRI	VI
NCut	0.40	0.76	2.39
SC	0.44	0.77	2.24
WSC	0.44	0.77	2.21
Yu-PFE	0.52	0.79	1.91
Ours	0.53	0.79	2.10

TABLE I: Comparison of Normalized Cut (NCut), spectral clustering (SC), weighted spectral clustering (WSC), Yu et al. implementation of PFE (Yu-PFE), and our efficient PFE method (Ours) on BSDS5000. Best results for each performance measure highlighted in bold.

V. CONCLUSION

Piecewise Flat Embeddings, originally proposed in [1], provide powerful data representations that can be used for clustering and image segmentation. However, the description of the original algorithm can be improved to incorporate two efficiencies: the reduction of the Split-Bregman iteration to work on more compact matrices, and the use of a preconditioned conjugate gradient solver to rapidly solve the linear



Fig. 1: Selection of BSDS500 training images with PFE-based segmentations for various embedding dimensions d .

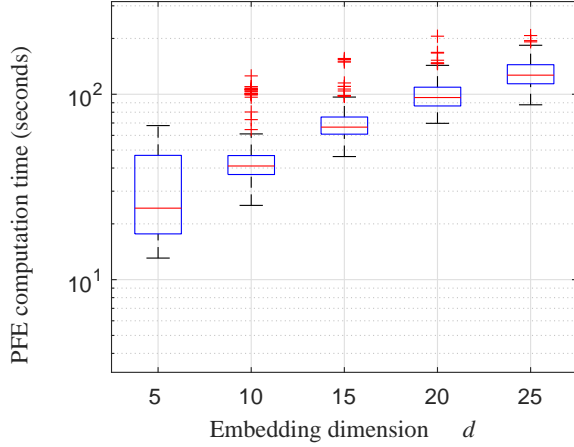


Fig. 2: Box plots of computation times required to efficiently compute PFE on BSDS500 images. Red +’s indicate outliers.

least squares problem at the heart of each inner loop. We showed that with these efficiencies, we can replicate the image segmentation experiment performed in [1] in a manner that yields similar performance measures but only requires 0.5–2 minutes per image instead of the 15 minutes previously reported.

CODE

A prototype implementation of our algorithm for efficiently computing PFE is available at MATLAB Central (<http://www.mathworks.com/matlabcentral/>) under File ID #59763.

REFERENCES

- [1] Y. Yu, C. Fang, and Z. Liao, “Piecewise flat embedding for image segmentation,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1368–1376. 1, 2, 3, 4
- [2] M. Belkin and P. Niyogi, “Laplacian eigenmaps for dimensionality reduction and data representation,” *Neural computation*, vol. 15, no. 6, pp. 1373–1396, 2003. 1
- [3] T. Goldstein and S. Osher, “The split Bregman method for L1-regularized problems,” *SIAM journal on imaging sciences*, vol. 2, no. 2, pp. 323–343, 2009. 1, 2
- [4] R. Lai and S. Osher, “A splitting method for orthogonality constrained problems,” *Journal of Scientific Computing*, vol. 58, no. 2, pp. 431–449, 2014. 2
- [5] R. Barrett, M. W. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, 1994, vol. 43. 3
- [6] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, “Contour detection and hierarchical image segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 5, pp. 898–916, 2011. 3